Mobile Networks and Applications 7, 407–418, 2002


# The Roma Personal Metadata Service

EDWARD SWIERK, EMRE KICIMAN, NATHAN C. WILLIAMS, TAKASHI FUKUSHIMA *, HIDEKI YOSHIDA **, VINCE LAVIANO and MARY BAKER
*Stanford University, Computer Science Department, Stanford, CA, USA*
E-mail: roma@mosquitonet.stanford.edu

**Abstract.** People now have available to them a diversity of digital storage facilities, including laptops, cell phone address books, handheld devices, desktop computers and web-based storage services. Unfortunately, as the number of personal data repositories increases, so does the management problem of ensuring that the most up-to-date version of any document in a user's personal file space is available to him on the storage facility he is currently using. We introduce the Roma personal metadata service to make it easier to locate current versions of personal files and ensure their availability across different repositories. This centralized service stores information *about* each of a user's files, such as name, location, timestamp and keywords, on behalf of mobility-aware applications. Separating out these metadata from the data respositories makes it practical to keep the metadata store on a highly available, portable device. In this paper we describe the design requirements, architecture and current prototype implementation of Roma.

**Keywords:** metadata, mobile computing, distributed databases, distributed data storage, personal systems, data synchronization

## 1. Introduction

As people come to rely more heavily on digital devices to work and communicate, they keep more of their personal files – including email messages, notes, presentations, address lists, financial records, news clippings, music and photographs – in a variety of data repositories. Since people are free to switch among multiple heterogeneous devices, they can squirrel away information on any device they happen to be using at the moment, as well as on an ever-broadening array of web-based storage services. For example, a businessperson wishing to record a travel expense could type it into his laptop, scribble it into his personal digital assistant, or record it in a web-based expense tracking service.

One might expect this plethora of storage options to be a catalyst for personal mobility [13], enabling people to access and use their personal files wherever and whenever they want, while using whatever device is most convenient to them. Instead, it has made it harder for mobile people to ensure that up-to-date versions of files they need are available on the current storage option of choice. This is because contemporary file management tools are poor at handling multiple data repositories in the face of intermittent connectivity. There is no easy way for a user to determine whether a file on the device he is currently using will be accessible later on another device, or whether the various copies of that file across all devices are up-to-date. As a result, the user may end up with many out-of-date or inconsistent copies of the same file scattered on different devices.

* Visiting from Kobe Steel Ltd., Electronics Research Laboratory, Kobe, Japan.

** Visiting from Toshiba Corporation, Corporate R&D Center, Kawasaki, Japan.

There are several approaches a user can take to solve the problems caused by storing files across several data repositories.

*Centralized file storage.* The simplest approach is for a user simply to store all of his files on the same device, thus sidestepping completely the problems of multiple storage repositories. This approach is not for everyone. Indeed, storing all of one's personal information – from private diaries to business records – on the same device may be considered too risky by some. But for others, the simplicity may be appealing.

A user can store all of his files on a PC at home or a workstation at work, and access them remotely from whatever device he happens to have available at the moment. To accomplish this, his system administrator could set up a web server or a network file system server such as NFS [16], which would let him view and manipulate the file repository on the client device via a network, using existing applications. However, the other, more challenging requirement is widespread, fast and inexpensive network access. Without it, the user may be tempted to store copies of some files on locally-accessible devices, thus defeating the advantages of the centralized approach. While such network access may exist someday, today we are unfortunately stuck choosing between fast, inexpensive local networks and slow, expensive global ones. Centralized file storage on a stationary server is therefore impractical.

Alternatively, a user can store all of his files on a portable device and simply carry that wherever he goes. The device must be small enough to carry everywhere and power-efficient enough to run for a day or so between battery charges. It must be capable of connecting to other devices like a laptop or PC for applications that require a larger display and input device. Although today's handheld devices meet many of

these requirements, their storage capabilities come up short. The continuing development of ever-more-demanding applications ensures that users will require physically large devices to store and manipulate their files for the forseeable future.

Other difficulties with the centralized approach include performance bottlenecks and potential reliability problems, due to the existence of a single point of failure in the central server.

*Distributed file storage.* A more sophisticated approach is for a user to store his personal files on different physical devices, and to use some mechanism to link them together so that they behave as one virtual device. By spreading files over several devices, the user can overcome many of the deficiencies of the centralized approach, such as slow or disconnected networks, device portability constraints, performance bottlenecks and reliability problems. However, the coupling between devices may still prove problematic for users who are loath to mix their private personal files with their professional files.

With a distributed file system a user can access his files through the network, using existing applications on any supported client device. Unlike a network file system, a distributed file system such as AFS [9] maintains storage on a set of servers that work together, functioning as a single virtual server while offering high performance and reliability. The Coda system [11] adds special support for client devices that operate while disconnected from the network. These multi-user, enterprise-scale systems generally support only higher-end devices like workstations, PCs and laptops. Therefore, users often maintain completely separate data repositories for smaller, more specialized devices.

Peer-to-peer file synchronization tools [2], like rsync [20] and HotSync [18], also manage files on multiple devices, but take a different tack from that of distributed file systems. Rather than attempting to unify several devices into a single virtual server, file synchronization tools let the user explicitly copy files between pairs of devices. Then the user can periodically invoke a synchronization operation that propagates changes made to files on one device to the copies on another device. While this kind of system is well-suited to pairs of small devices that are only occasionally connected to a network, it does not solve the problems of tracking multiple versions of files across many repositories.

The ideal solution would unify and centralize disparate file storage repositories, while preserving the flexibility of the distributed file storage approach. Users should be free to copy files to any device to ensure that the files can be found there later – personal financial records on the home PC, digital audio files in the car, phone numbers on the cell phone – without having to remember which copies reside on which devices and what copy was modified when.

Our system, Roma, provides an available, centralized repository of metadata, or information *about* a single user's files. The metadata format includes sufficient information to enable tracking each file across multiple file stores, such as a name, timestamp, and URI or other data identifier. A user's metadata repository may reside on a device that the user carries along with him (metadata records are typically compact enough that they can be stored on a highly portable device), thus ensuring that metadata are available to the user's local devices even when wide area network connectivity is intermittent. To maintain compatibility with existing applications, synchronization agents periodically scan data stores for changes made by legacy applications and propagate them to the metadata repository.

Related to the problem of *managing* versions of files across data repositories is the problem of *locating* files across different repositories. Most file management tools offer hierarchical naming as the only facility for organizing large collections of files. Users must invent unique, memorable names for their files, so that they can find them in the future, and users must arrange those files into hierarchies, so that related files are grouped together. Having to come up with a descriptive name on the spot is an onerous task, given that the name is often the only means by which the file can later be found [15]. Arranging files into hierarchical folders is cumbersome enough that many users do not even bother, and instead end up with a single "Documents" folder listing hundreds of cryptically named, uncategorized files. This problem is compounded when files need to be organized across multiple repositories.

Roma metadata include fully extensible attributes that can be used as a platform for supporting other methods of organizing and locating files. While our current prototype clients take advantage of such attributes for simple querying, several projects have explored the use of attribute-based naming to locate files in either single or multiple repositories [4,7] using arbitrarily complex queries.

The rest of this paper describes Roma in detail. We begin by outlining the requirements motivating our design. In subsequent sections we detail the architecture and current prototype implementation of Roma, as well as some key issues that became apparent while designing the system. These sections are followed by a survey of related work and a discussion of some possible future directions for this work.

## 2. Motivation and design requirements

To motivate this work, consider the problems faced by Jane Mobile, techno-savvy manager at ABC Widget Company, who uses several computing devices on a regular basis. She uses a PC at work and another at home for editing documents and managing her finances, a handheld organizer for storing her calendar, a laptop for working on the road, and a cell phone for keeping in touch. In addition, she keeps a copy of her calendar on a web site so it is always available both to herself and to her co-workers, and she frequently downloads the latest stock prices into her personal finance software.

Before dashing out the door for a business trip to New York, Jane wants to make sure she has everything she will need to be productive on the road. Odds are she will forget something, because there is a lot to remember:

- *I need to bring the latest price list for blue fuzzy widgets, which is probably somewhere on my division's web site or on the group file server, or maybe on my laptop.* Jane does not want to click her way through window after window just to locate a single document; she should be able go to one place to browse or search all the documents she considers relevant, regardless of where they are actually stored. Even though the file server and the web site are completely outside her control, Jane would like to use the same tools that she uses to locate documents on her own storage devices.
- *I have to make some changes to that presentation I was working on yesterday. Did I leave the latest copy on my laptop or on my PC at home?* If Jane copies an outdated version to her laptop, she may cause a write conflict that will be difficult to resolve when she gets back. She just wants to grab the presentation without having to check both computers to figure out which version is the more recent one.
- *I promised my client I'd bring along the specifications document for blue fuzzy widgets – I think it's called BFWidgetSpec.doc, or is it SpecBluFuzWid.doc?* If Jane could do a keyword search over all her documents (regardless of which applications she used to create them) and over all her devices at once, she would not have to remember what the file is called, which directory contains it, or on which device it is stored.
- *I want to work on my expense report on the plane, so I'll need to bring along my financial files.* Like most people, Jane does not have the time or patience to arrange all her documents into neatly labeled directories, so it's hard for her to find groups of related files when she really needs them. More likely, she has to pore over a directory containing dozens or hundreds of files, and guess which ones might have something to do with her travel expenses.

To summarize, the issues illustrated by this example are the need for a tool a user can rely on to locate files stored on any of his devices or storage repositories; the difficulty of keeping track of multiple versions of a file across different devices; and the unfortunate dependence on filenames and directories for identifying and grouping files together.

These issues lead us to a set of architectural requirements for Roma. Our solution should be able to:

1. *Make information about all of the user's personal files always available to applications and to the user.*
2. *Associate with each file (or file copy) a set of standard attributes*, including version numbers or timestamps to help synchronize file replicas and preempt many write conflicts.
3. *Allow the attribute set to be extended by applications and users*, to include such attributes as keywords for searching, categories for browsing related files, digests or thumbnails for previewing file content, and parent directories for traditional hierarchical naming (where desired). This information can be used to develop more intuitive methods to organize and locate files.
4. *Track files stored on data repositories outside the user's control.* A user may consider a certain file as part of his personal file space even if he did not directly create or maintain the data. For example, even though the user's bank account balances are available on a web site controlled and maintained by the bank, he should be able to organize, search and track changes to these data just like any other file in his personal space.
5. *Track files stored on disconnected repositories and offline storage media.* Metadata can be valuable even if the data they describe are unavailable. For example, the user may be working on a disconnected laptop on which resides a copy of the document that he wants to edit. Version information lets him figure out whether this copy is the latest, and if not, where to find the most recent copy upon reconnection. Alternatively, if the laptop is connected on a slow network, he can use metadata (which are often smaller in size than their associated file) to find which large piece of data needs to be pulled over the network.

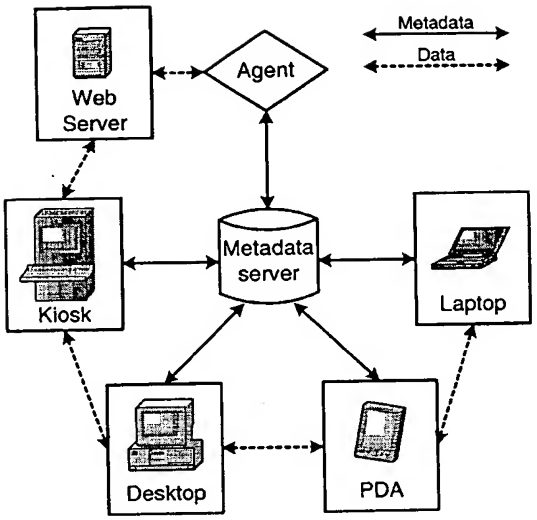


Figure 1. The Roma architecture. Applications running on each of a user's devices communicate with a centralized personal metadata server. Agents track changes made to data stored on devices outside the user's control, such as the web server in this diagram, and make appropriate updates to the metadata server.

## 3. Architecture

At the core of the Roma architecture (illustrated in figure 1) is the *metadata server*, a centralized, potentially portable service that stores information about a user's personal files. The files themselves are stored on autonomous data repositories, such as traditional file systems, web servers and any other device with storage capability. Roma-aware applications query

```
<metadata>
  <uid>123456789</uid>
  <name>
    Blue Fuzzy Widget specifications
  </name>
  <location>
    <protocol>http</protocol>
    <host>anthill.stanford.edu</host>
    <path>
      /projects/bluestuff/BFWidgetSpec.doc
    </path>
  </location>
  <version>12</version>
  <attribute>
    <key>type</key>
    <value>
       Microsoft Office Document
    </value>
  </attribute>
  <attribute>
    <key>keyword</key>
    <value>blue</value>
  </attribute>
  <attribute>
    <key>author</key>
    <value>Jane Mobile</value>
  </attribute>
</metadata>
```

Figure 2. A typical metadata record. This record describes an instance of Jane's specifications document that resides on host `anthill.stanford.edu`. The `uid`, `name`, `location` and `version` tags are standard elements; `attribute` tags contain optional, domain-specific attributes.

the metadata server for file information, and send updates to the server when the information changes. Applications obtain file data directly from data repositories. Agents monitor data stores for changes made by Roma-unaware applications, and update file information in the metadata server when appropriate.

Roma supports a decentralized replication model where all repositories store "first class" file replicas – that is, all copies of a file can be manipulated by the user and by applications. To increase availability and performance, a user can copy a file to local storage from another device, or an application can do so on the user's behalf. Roma helps applications maintain the connection between these logically related copies, or *instances*, of the file by assigning a unique file identifier (UID) that is common to all of its instances. The file identifier can be read and modified by applications but is not normally exposed to the user.

Once the file is copied, the contents and attributes of each instance can diverge. Thus, Roma keeps one *metadata record* for each file instance. A metadata record is a tuple composed of the UID, one or more data locations, a version number and optional, domain-specific attributes. Figure 2 shows a typical metadata record.

The *data location* specifies the location of a file instance as a Universal Resource Identifier (URI) [22]. Files residing on the most common types of data repositories can be identified using existing URI schemes, such as `http:` and `ftp:` for network-accessible servers and `file:` for local file systems. When naming removable storage media, such as a CD-ROM or a Zip disk, it is important to present a human-understandable name to the user (possibly separate from the media's native unique identifier, such as a floppy serial number).

The *version number* is a simple counter. Whenever a change is made to a file instance, its version number is incremented.

Roma-aware applications can supplement metadata records with a set of optional *attributes*, stored as name/value pairs, including generic attributes such as the size of a file or its type, and domain-specific attributes such as keywords, categories, thumbnails, outlines or song titles.

These optional attributes enable application user interfaces to support new modes of interaction with the user's file space, such as query-based interfaces and browsers. In section 4.2 we describe some autonomous agents that we have built to scan files in the user's space automatically and add attributes to the metadata server based on the files' contents. Section 6 briefly describes Presto, a system developed by the Placeless Documents group at Xerox PARC that allows users to organize their documents in terms of user-defined attributes. The user interaction mechanisms developed for Presto would mesh well with the centralized, personal metadata repository provided by Roma.

### *3.1. Metadata server*

The metadata server is a logically centralized entity that keeps metadata information about all copies of a user's data. Keeping this metadata information centralized and separate from the data stores has many advantages:

- Centralization helps avoid write conflicts, since a single entity has knowledge of all versions of the data in existence. Some potential conflicts can be prevented before they happen (before the user starts editing an out-of-date instance of a file) rather than being caught later, when the files themselves are being synchronized.
- Centralization allows easier searching over all of a user's metadata because applications only have to search at a single entity. The completeness of a search is not dependent on the reachability of the data stores. In contrast, if metadata were distributed across many data stores, a search would have to be performed at each data store. While this is acceptable for highly available data repositories connected via high-bandwidth network, it is cumbersome for data stores on devices that need to be powered on, plugged in, or dug out of a shoebox to be made available.
- Separation of the metadata from the data store allows easier integration of autonomous data stores, including legacy and third-party data stores over which the user has limited control. Storing metadata on a server under the user's control, rather than on the data stores with the data, eliminates the need for data stores to be Roma-aware. This greatly eases the deployability of Roma.

- Separation also makes it feasible to impose a personalized namespace over third-party or shared data. A user can organize his data in a manner independent of the organization of the data on the third-party data store.
- Separation enables applications to have some knowledge about data that are currently inaccessible, either because the data store is offline or because it speaks a foreign protocol.

The main challenge in designing a centralized metadata server is ensuring that it is always available despite intermittent network connectivity. Section 5.2 describes one solution to this problem, which is to host the metadata server on a portable device kept close to the user. Since metadata tend to be significantly smaller than the data they describe, it is feasible for users to take their metadata server along with them when they disconnect from the network.

### *3.2. Data stores*

A data store is any information repository whose contents can somehow be identified and retrieved by an application. Roma-compatible data stores include not only traditional file and web servers, but also laptops, personal digital assistants (PDAs), cell phones, and wristwatches – devices that have storage but cannot be left running and network-accessible at all times due to power constraints, network costs, and security concerns – as well as offline storage media like compact discs and magnetic tapes. Information in a data store can be dynamically generated (for example, current weather conditions or bank account balances). Our architecture supports:

- data stores that are not under the user's control;
- heterogeneous protocols (local file systems, HTTP, FTP, etc.). There are no restrictions on the protocols supported by a data store;
- data stores with naming and hierarchy schemes independent of both the user's personal namespace and other data stores.

In keeping with our goal to support legacy and third-party storage facilities, data stores do not have to be Roma-aware. There is no need for direct communication between data stores and the metadata server. This feature is key to ensuring the deployability of Roma.

### *3.3. Applications*

In Roma, applications are any programs used by people to view, search and modify their personal data. These include traditional progams, such as text editors, as well as personal information managers, web-based applications, and special-purpose Internet appliances. Applications can be co-located with data sources; for example, applications running on a desktop computer are co-located with the computer's local file system.

Roma-aware applications have two primary responsibilities. The first is to take advantage of metadata information already in the repository, either by explicitly presenting relevant metadata to the user or by automatically using metadata to make decisions. For example, an application can automatically choose to access the "nearest" or latest copy of a file.

The application's second responsibility is to inform the metadata server when changes are made to the data that affect the metadata. At the very least, this means incrementing the version number when a change has been made (for synchronization purposes), but can also include updating domain-specific metadata. We are investigating how often updates need to be sent to the metadata server to balance correctness and performance concerns.

While applications should be connected to the metadata server during use, they are not necessarily well connected to all data stores; they may be connected weakly or not at all. For example, an application might not speak the protocol of a data store, and thus might be effectively disconnected from it. Also, a data store itself may be disconnected from the network.

### *3.4. Agents*

Roma agents are software programs that run on behalf of the user, without requiring the user's attention. These agents can perform many tasks, including:

- providing background updates of metadata on behalf of the user (for example, updating metadata in response to changes made by non-Roma-aware applications);
- warning a user when he is about to edit an out-of-date version of a document;
- hoarding files in preparation for disconnected operation;
- making timely backups of information across data stores; and
- tracking third-party updates on autonomous data stores like web servers.

We have found it useful to classify agents based on the nature of their interactions with the metadata server:

**Read-only.** Some agents require only read access to metadata. An agent that warns the user that he is about to edit a stale instance of a document is an example of a read-only agent.

**Write-only.** This class of agents only writes to the metadata server, but does not need to read information from it. Agents that scan documents for changes made by third parties or non-Roma-aware applications can fall into this category.

**Read/write.** Other agents require both read and write access to a metadata server. Agents that make backups or hoard files require both read access (to decide what files to copy) and write access (to inform the metadata server of the new copies of the files).

Agents can be run anywhere on a user's own devices or on cooperating infrastructure. The only limitation on an agent's

location is that the agent must be able to access both the relevant data stores and the metadata server. Note that the use of a portable metadata server precludes some kinds of agents from running while the metadata server is disconnected from the rest of the network; section 5.2 describes a way to mitigate the effects of this disconnection on agents.

### 3.5. Examples

To illustrate how Roma supports a user working with files replicated across several storage devices, let us revisit Jane Mobile and consider what a Roma-aware application does in response to Jane's actions.

The action of copying a file actually has two different results, depending on her intent, and the application should provide a way for her to distinguish between the two:

- *She makes a file instance available on a different repository* (in preparation for disconnected operation, for example). The application contacts the metadata server, creates a new metadata record with the same file identifier, copies all attributes, and sets the data location to point to the new copy of the file.
- *She copies a file to create a new, logically distinct file based on the original.* The application contacts the metadata server, creates a new metadata record with a new file identifier, copies all attributes, and sets the data location to point to the new copy of the file.

Other actions Jane may take:

- *She opens a file for updating.* The application contacts the metadata server, and checks the version number of this instance. If another instance has a higher version number, the application warns Jane that she is about to modify an old version, and asks her if she wants to access the latest version or synchronize the old one (if possible).
- *She saves the modified file.* The application contacts the server, increments the version number of this instance, and updates any attributes, such as the file's size. As described in section 5.1, a write conflict may be detected at this point if the version number of another instance has already been incremented.
- *She brings a file instance up to date by synchronizing it with the newest instance.* The application contacts the server, finds the metadata record with the highest version number for this file, and copies all attributes (except the data location) to the current instance.

### 3.6. Limitations

The Roma architecture meets our requirements for providing metadata about a user's personal documents, even when those documents are on different devices or devices that are disconnected or powered off. It provides a mechanism for tracking update information for multiple copies of documents, associating extended attributes with documents, and searching for documents based on these metadata.

```
PDFSServerIF server =
  (PDFSServerIF) loader.
  getService("pdfs.server.PDFSServer");
Metadata q = new Metadata ();
q.setName("Blue Fuzzy Widget " + "specifications");
server.query(q);
```

Figure 3. A sample Java client query. `server` is the interface to a Ninja iSpace service. `q` is a query that specifies the content of the <name> metadata tag.

Although this architecture meets our requirements, some important issues remain: ensuring that the metadata store is available to the user's applications and to third-party synchronization agents, and revising applications to take advantage of the metadata store to aid the user in synchronizing and locating files. These issues are discussed in sections 5.2 and 5.4, respectively.

## 4. Implementation

In this section we describe the current status of our prototype Roma implementation.

### 4.1. Metadata server

We have implemented a prototype metadata server that supports updates and simple queries, including queries on optional attributes. It is written in Java as a service running on Ninja [8], a toolkit for developing highly available network services. Metadata are stored in an XML-based format, and we use XSet [23], a high performance, lightweight XML database, for query processing and persistence.

Java-based clients can query and update the metadata server through a native RMI interface. Figure 3 shows a snippet of code from a typical client.

Clients written in other languages, such as C, can use an XML-over-HTTP gateway interface. Figure 4 shows a query interaction between a client and the gateway.

We have also implemented a proof-of-concept portable metadata server. Though the metadata server itself requires a full Java environment to operate, we have implemented a simple mechanism to migrate a metadata repository between otherwise disconnected computers using a PDA as a transfer medium. As a user finishes working on one computer, the metadata repository is transferred onto his PDA. The next time he begins using a computer, the metadata repository is retrieved from the PDA. In this way, though the prototype metadata server itself is not traveling, the user's metadata are always accessible, regardless of the connectivity between the user's computer and the rest of the world.

### 4.2. Data stores

Currently, the data stores we support are limited to those addressable through URIs. Our applications can currently access data stores using HTTP and FTP, as well as files accessible via a standard file system interface.

```
POST /ISPACE/pdfs.client.http.RMAPServer
  ?query HTTP/1.1
<?xml version="1.0"?>
<metadata>
<name>
  Blue Fuzzy Widget specifications
</name>
</metadata>

HTTP/1.1 200 OK

<?xml version="1.0"?>
<metadata>
<uid>123456789</uid>
<name>
  Blue Fuzzy Widget specifications
</name>
<location>
...
```

Figure 4. A sample XML-over-HTTP query interaction. A client establishes a HTTP connection to the gateway and issues a POST request, specifying a query operation in the URL suffix. The request body designates the content of the <name> metadata tag. Corresponding metadata records are returned to the client in the HTTP response.

### *4.3. Agents*

We have built an example of a write-only agent to synchronize metadata in the Roma server in response to actions made by non-Roma-aware applications. Our agent performs periodic scans of files under its management, searching for file updates. Upon detecting an update, it uses a set of file type-specific analyzers to collect new metadata from the updated document.

Our agent is written in C, and speaks with the metadata server through its XML-over-HTTP interface. We have currently implemented file analyzers to extract image size and color depth information from GIF and JPEG image files; bibliographic information such as title and author from Word and LaTeX documents; and library dependency information from ELF-formatted binary files. In particular, the ELF analyzer promises to be useful when calculating a set of programs to hoard.

Other agents, such as the read-only "warning" agent, and the hoarding and backup agents remain future work.

### *4.4. Applications*

We have implemented three Roma-aware applications. These applications allow users to view and manipulate their metadata and data from a variety of devices.

The first is a web-based metadata browser that provides hierarchical browsing of a user's personal data. The browser displays the names of data files, their version information, and the deduced MIME type of the file. In addition, if the file is accessible, the browser will present a link to the file itself. We have also written a proxy to enable "web clipping" of arbitrary web content into the user's personal file space, as displayed in figure 5.

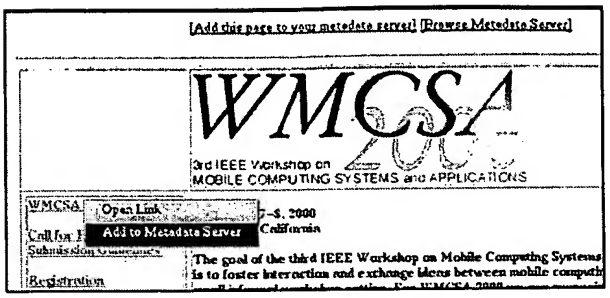


Figure 5. A screen shot of the web clipper proxy. As the user browses the web, the proxy adds links on the fly, allowing the user to browse the metadata server and add pages to his personal file space.

Our second application is a set of command-line tools. We have written Roma-aware `ls` and `locate` commands to query a metadata server, a `get` command to retrieve the latest version of a file from remote data stores, and `import`, a utility to create metadata entries for files on a local data store.

We have also implemented a proof-of-concept PDA application. Built using a Waba VM and RMILite [3,21], our PDA application can query and view the contents of a metadata server. Currently, the PDA application is limited to browsing metadata and cannot display the file data themselves.

Our applications extend the basic Roma metadata record with a format attribute to describe the data format of files. If available, our command-line tools use the Unix `magic` command to determine the data format. Our web clipper determines the data format based on the MIME type of the file.

## 5. Design issues and future work

In this section we describe some of the issues and design decisions encountered so far in our work with Roma, along with some of the work that remains for us to do.

### *5.1. Why "personal"?*

One important design issue in Roma is the scope of the types of data it supports. There are several reasons behind our choice to support only personal files, rather than to tackle collaboration among different users as well, or to attempt to simplify system administration by handling distribution of application binaries and packages.

First, restricting ourselves to personal files gives us the option of migrating the metadata server to a personal, portable device that the user carries everywhere, to increase its availability. This option is described in more detail in the next section.

Second, it avoids a potential source of write conflicts – those due to concurrent modifications by different users on separate instances of the same file. Such conflicts are often difficult to resolve without discussion between the two users.

However, single users can still make concurrent modifications to their own files [1], potentially causing conflicts. Conflicts can also result from modifications by third parties

working on a user's behalf, such as an email transfer agent appending a new message to the user's inbox while the user deletes an old one. However, these conflicts can often be resolved automatically using knowledge about the application, such as the fact that an email file consists of a sequence of independent messages. Another way a user can create conflicts himself is by concurrently executing applications that access the same document. Avoiding this behavior is usually within the control of the user, and any resulting conflicts do not require coordination among multiple users for resolution. We are investigating the use of version vectors to store more complete and flexible version information [14].

Third, it lets us exploit the fact that users are much better at predicting their future needs for their personal files than for other kinds of files [5].

Fourth, it lets us support categories, annotations and other metadata that are most meaningful to a single person rather than a group.

Finally, we believe there is a trend toward specialized applications tailored for managing other types of files:

- Groupware systems like the Concurrent Versioning System (CVS), ClearCase, Lotus Notes and Microsoft Outlook impose necessary structure and order on access to *shared data with multiple writers*. Email is often sufficient for informal collaboration within a small group.
- Tools like the RedHat Package Manager (RPM) and Windows Update are well suited for distributing *system-oriented data* such as application packages, operating system components, and code libraries. These tools simplify system administration by grouping related files into packages, enforcing dependencies, and automatically notifying the user of bug fixes and new versions of software.
- The web has become the best choice for distributing *shared data with many readers.*

Since these applications handle system data, collaborative projects and shared read-mostly data, we believe that the remaining important category of data is personal data. We, thus, focus on handling this category of data in Roma.

### 5.2. *Ensuring availability of metadata*

Since our overarching goal is to ensure that information about the user's files is always available to the user, we need to make the system robust in the face of intermittent or weak network connectivity – the very situations that underscore the need for a metadata repository in the first place.

Our approach is to allow the user to keep the metadata server in close physical proximity, preferably on a highly portable device that he can always carry, as described in the next section. Wireless network technologies will soon make "personal-area networks" a reality, thus enabling a Roma device to communicate with other computing and communication devices in its vicinity. It is not hard to imagine a server embedded in a cell phone or a PDA, with higher availability and better performance than a remote server in many situations. Metadata can then be close and accessible to whatever application, device or data store the user is currently accessing.

The main difficulty with storing metadata on a portable server is making the metadata available to agents that act on behalf of the user and modify data in the user's personal file space. If the network is partitioned and the only copy of the metadata is with the user, how does such an agent read or modify them? In other words, we need to ensure that metadata are available to remote entities as well as to the user's local applications.

One solution is to cache subsets of the metadata from the metadata server in multiple locations. If the metadata server currently resides on the user's handheld device, a replica residing on a stationary, network-connected server can provide access to third parties. This naturally raises the issues of synchronizing and handling update conflicts among the metadata replicas.

However, our hypothesis is that updates made to the metadata by third parties rarely conflict with user updates. For example, a bank's web server updates a file containing the user's account balances, but the user himself rarely updates this file. In this case, the agent monitoring the web server for updates is a write-only agent, and is the sole writer of the metadata record for the account balances file. Studying metadata access patterns such as this one in greater depth is part of our future work in evaluating Roma.

### 5.3. *How much storage does Roma need?*

Estimating the storage requirements of the metadata server is key to justifying the usefulness of the Roma system. We need to ensure that the amount of metadata needed for a typical user's files does not exceed the storage capacity of a practical portable device. Furthermore, even given a sufficiently large storage device, if the metadata are as large as the data they describe, then there is little benefit in building a personal metadata service. Thus, there are two important measurements to consider: the absolute amount of storage required for the metadata server, and the ratio of the size of the metadata records to the size of the files they describe.

While the Roma prototype has not been used widely enough for us to take measurements, a preliminary survey of the home directories of several research group members gives us some numbers from which we can infer an estimate. (Each member of the research group ran the Unix commands `find ~ | wc` to count the files in his home directory, and `du -s ~` to compute the sum of the files' sizes.)

The number of files in a user's personal file space ranges from 10,000 to 100,000 and the average file size is 25,000 to 250,000 bytes. Metadata records storing the kinds of information shown in figure 2 might weigh in at 1,000 bytes on average. Then in the worst case the metadata server would require approximately 100 megabytes of storage, and the ratio of metadata record size to file size would be 1 to 25.

Since a 100-megabyte metadata repository would easily fit on a flash memory card or miniature hard drive, Roma is practical to implement today. In addition, it is considerably less expensive to build a device capable of running a metadata server with reasonable performance than one storing data 25 times larger.

### *5.4. Making applications Roma-aware*

Making applications Roma-aware is the biggest challenge in realizing Roma's benefits of synchronization and file organization across multiple data stores. To gain the most benefit, application user interfaces and file input/output routines must be adapted to use and update information in the metadata store. We have several options for extending existing applications to use Roma or incorporating Roma support into new applications.

Our first option is to use application-specific extension mechanisms to add Roma awareness to legacy applications. For example, we implemented a Roma-aware proxy to integrate existing web browsers into our architecture. Roma add-in modules could be written for other applications that have extension APIs, such as Microsoft Outlook, or for open-source applications that can be modified directly.

Our second option is to layer Roma-aware software beneath the legacy application. Possibilities include modifying the C library used by applications to access files, or writing a Roma-aware file system. This option does nothing to adapt the application's user interface, but it can provide some functionality enhancements such as intelligent retrieval of updated copies of files.

The third option, which we have implemented, is to use agents to monitor data edited by legacy applications in the same way we monitor data repositories not under the user's control. These agents ensure that the metadata at the server are kept up-to-date with changes made by legacy applications. They can also perform other tasks to enhance non-Roma-aware applications, such as synchronizing local files with other data repositories, or notifying the user that updated file instances are available.

Beyond choosing the most appropriate method to extend an application to use Roma, the bulk of the programming effort is in modifying the application's user interface and communicating with the metadata store. Our current prototype provides simple, generic HTTP and Java RMI interfaces to the metadata store, through which applications pass XML-formatted objects. Platform- or domain-specific Roma libraries could offer much richer support to application developers, including both user interface and file I/O components, to help minimize the programming effort. For example, a Roma library for Windows could offer a drop-in replacement for the standard "file explorer" components, so that adapting a typical productivity application would involve making a few library API calls rather than developing an entirely new user interface.

### *5.5. Identifying file instances*

Our current Roma implementation uses a URI to identify the file instance corresponding to a particular metadata record. Unfortunately, using URIs as file identifiers is problematic: a Roma user could end up with several inconsistent metadata records for the same file instance, or even worse, the user could delete the only copy of a file because the metadata server shows that a backup instance exists at a "different" location.

These problems stem from the fact that URIs are not intended for one-to-one identification of file instances. On many systems, a file instance can be identified by more than one URI, due to aliases and links in the underlying file system or multiple network servers providing access to the same files. For example, the file identified by `ftp://gunpowder/pub/paper.ps` can also be identified as `http://gunpowder/ftp/pub/paper.ps`, since the public FTP directory is also exported by an HTTP server.

Currently we rely on applications and agents to detect and handle cases where multiple URIs refer to the same file, but this places a large burden on application writers. In the future, Roma must address this problem more systematically.

## 6. Related work

### *6.1. File systems*

Helping users access data on distributed storage repositories is an active area of research. The primary characteristic distinguishing our work from distributed file systems, such as Coda [11], OceanStore [12], and Bayou [14], is our focus on providing highly available metadata rather than ensuring availability of the data themselves. In fact, the benefits of Roma can be attained by using it alongside each of these systems.

The Coda distributed file system seeks to allow users to remain productive during periods of weak or no network connectivity. To accomplish this, it caches file data according to user preferences in anticipation of periods of disconnection or weak connectivity. From the user's point of view, files in uncached directories are simply missing, which can be a jarring experience if the files were present just a few minutes ago. With Roma, having metadata available for all files allows the system to maintain the user's view of his personal file space and provide useful feedback when the data of an uncached file are unavailable, like "The file `/projects/bluestuff/BFWidgetSpec.doc` is currently unavailable; connect to Jane's Home PC (`anthill.stanford.edu`) to access this file." Coda could also be used to replicate the metadata repository itself, implementing the caching mechanism described in section 5.2.

The architecture of OceanStore is similar to that of Coda, but in place of a logically single, trusted server is a global data utility comprised of a set of untrusted servers whose owners earn a fee for offering persistent storage to other users. Weakly connected client devices can read from and write to

the closest available server; the infrastructure takes care of replicating and migrating data and resolving conflicts. As with Coda, in the absence of network connectivity, both data and metadata are unavailable for OceanStore-managed files. A user running Roma alongside OceanStore can benefit from having a consistent view of his file space even when no server is available to provide file data.

The Bayou system supports a decentralized model where users can store and modify their files in many repositories that communicate peer-to-peer to propagate changes. A given repository might contain only a subset of the files in the user's personal file space, at varying degrees of freshness, depending on how long it has been since the repository was in contact with the others. Again, having Roma available to provide an always-current set of file metadata can help the user and his applications deal with missing or out-of-date files.

SyncML [19] is a lightweight, XML-based data synchronization protocol, tailored to work on resource-constrained devices and low-bandwidth networks. It is not a complete system but rather a protocol on which a synchronization system could be built. Roma could be used alongside a SyncML-based system just as described above for Coda or Bayou. Roma could also be implemented using SyncML to transport metadata between clients and the metadata server.

The Presto system [4] focuses on enabling users to organize their files more effectively. The Presto designers have built a solution similar to Roma that associates with each of a user's documents a set of properties that can be used to organize, search and retrieve files. This work does not specifically address tracking and synchronizing multiple copies of documents across storage repositories, nor does it ensure that properties are available even when their associated documents are inaccessible. However, the applications they have developed could be adapted to use the Roma metadata server as property storage.

Both Presto and the Semantic File System [7] enable legacy applications to access attribute-based storage repositories by mapping database queries onto a hierarchical namespace. Presto achieves this using a virtual NFS server, while the Semantic File System integrates this functionality into the file system layer. Either mechanism could be used with Roma to provide access to the metadata server from Roma-unaware applications.

The Elephant file system [17] employs a sophisticated technique for tracking files across both changes in name and changes in inode number.

### *6.2. Other metadata service applications*

The Roma metadata server is designed for managing personal data stored on a number of heterogeneous devices. Roma allows users to view information about all of their data, regardless of the capability of any particular device to access and view their data. A related project, the Dataheap, combines a centralized metadata server with a transformation service [10] that automatically converts a document from its native format to one accessible on the user's current device. The Dataheap is being developed as part of the Interactive Workspaces project at Stanford University [6].

The Dataheap is being used in a conference room environment to manage data access from networked heterogeneous devices, from large wall-sized displays to PDAs. Data can be stored on any semi-permanent device in the room; to keep a history of activities in the room, data originating from handheld or laptops is transferred to permanent file stores. Metadata in the Dataheap, like that of Roma, includes basic name, data format, and location information. Dataheap clients query their metadata server with a protocol similar to Roma's. The major difference is that when Dataheap clients query for information in a particular data format, the Dataheap server will attempt to convert data with otherwise-matching metadata into the requested data format.

In this environment, the Dataheap's attribute-based naming is used to provide a convenient, device-independent method of referencing information, while the transformational capabilities of the Dataheap assure that information is accessible from varied devices, though sometimes in a degraded format. We have already used the Dataheap to build a presentation management program, choreographing the display of slides and other documents across multiple heterogeneous devices. We are currently extending our use of metadata to keep data access histories in our interactive room, as well as using attributes to name the current "clipboard" contents in the room.

## 7. Conclusions

We have described a system that helps fulfill the promise of personal mobility, allowing people to switch among multiple heterogeneous devices and access their personal files without dealing with nitty-gritty file management details such as tracking file versions across devices. This goal is achieved through the use of a centralized metadata repository that contains information about all of the user's files, whether they are stored on devices that the user himself manages, on remote servers administered by a third party, or on passive storage media like compact discs. The metadata can include version information, keywords, categories, digests and thumbnails, and the format is completely extensible. We have implemented a prototype metadata repository, designing it as a service that can be integrated easily with applications. The service can be run on a highly available server or migrated to a handheld device so that the user's metadata are always accessible.

## Acknowledgements

The authors thank Doug Terry for his helpful advice throughout the project, and Petros Maniatis for his assistance with the figures. We also thank Andy Huang, Kevin Lai, Petros Maniatis, Mema Roussopoulos, and Doug Terry for their detailed review and comments on the paper. This work has been supported by a generous gift from NTT Mobile Communications Network, Inc. (NTT DoCoMo).

## References


[1] M.G. Baker, J.H. Hartman, M.D. Kupfer, K.W. Shirriff and J.K. Ousterhout, Measurements of a distributed file system, in: *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, Pacific Grove, CA (October 1991).

[2] S. Balasubramaniam and B.C. Pierce, What is a file synchronizer?, in: *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98)*, Dallas, TX (October 1998).

[3] M. Chen, M. Lakhamraju, E. Brewer and D. Culler, Jini/RMI/TSpace for small devices.

[4] P. Dourish, W.K. Edwards, A. LaMarca and M. Salisbury, Uniform document interactions using document properties, in: *Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology (UIST'99)*, Asheville, NC (November 1999).

[5] M. Ebling, Translucent cache management for mobile computing, Thesis, School of Computer Science, Carnegie Mellon University (1998).

[6] A. Fox, B. Johanson, P. Hanrahan and T. Winograd, Integrating information appliances into an interactive workspace, in: IEEE Computer Graphics and Applications 20(3) (May/June 2000).

[7] D.K. Gifford, P. Jouvelot, M.A. Sheldon and J.W. O'Toole, Jr., Semantic file systems, in: *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, Pacific Grove, CA (October 1991).

[8] S. Gribble, M. Welsh, E.A. Brewer and D. Culler, The MultiSpace: an evolutionary platform for infrastructural services, in: *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems (USITS'99)* (August 1999).

[9] J.H. Howard, An overview of the Andrew File System, in: *Proceedings of the USENIX Winter Technical Conference*, Dallas, TX (February 1988).

[10] E. Kıcıman and A. Fox, Using dynamic mediation to integrate COTS entities in a ubiquitous computing environment, in: *Proceedings of the Second International Symposium on Handheld and Ubiquitous Computing (HUC 2000)*, Bristol, England (September 2000).

[11] J.J. Kistler and M. Satyanarayanan, Disconnected operation in the Coda file system, in: *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, Pacific Grove, CA (October 1991).

[12] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells and B. Zhao, OceanStore: An architecture for global-scale persistent storage, in: *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, Cambridge, MA (November 2000).

[13] P. Maniatis, M. Roussopoulos, E. Swierk, K. Lai, G. Appenzeller, X. Zhao and M. Baker, The Mobile People Architecture, ACM Mobile Computing and Communications Review ($MC^2R$) (July 1999).

[14] K. Petersen, M.J. Spreitzer, D.B. Terry, M.M. Theimer and A.J. Demers, Flexible update propagation for weakly consistent replication, in: *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, Saint-Malo, France (October 1997).

[15] J. Raskin, *The Humane Interface* (Addison-Wesley, 2000).

[16] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh and B. Lyon, Design and implementation of the Sun Network File System, in: *Proceedings of the Summer 1985 USENIX Conference*, Portland, OR (June 1985).

[17] D.S. Santry, M.J. Feeley, N.C. Hutchinson, A.C. Veitch, R.W. Carton and J. Ofir, Deciding when to forget in the Elephant file system, in: *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, Charleston, SC (December 1999).

[18] S. Slack, Extending your desktop with Pilot, PDA Developer Magazine (September/October 1996).

[19] SyncML Consortium, SyncML specification, Version 1.0.1, `http://www.syncml.org/downloads.html`

[20] A. Tridgell and P. Mackerras, The rsync algorithm, Technical report TR-CS-96-05, Australian National University (1996).

[21] Wabasoft Inc., Wabasoft: Product overview, `http://www.wabasoft.com/products.shtml`

[22] World Wide Web Consortium, Naming and addressing: URIs, URLs, ..., `http://www.w3.org/Addressing/`

[23] B.Y. Zhao and A.D. Joseph, XSet: A lightweight database for Internet applications (May 2000) `http://www.cs.berkeley.edu/~ravenben/xset/html/papers.html`


**Edward Swierk** is a PhD candidate in the Computer Science Department at Stanford University. His interests include mobile computing, data management and personal systems. He received a BS degree in computer science in 1994 from Comell University, and an MS degree in 1999 from Stanford University.

**Emre Kiciman** is a PhD candidate in the Computer Science Department at Stanford University. He received a BS degree from the University of California at Berkeley in 1999. His research interests include software infrastructures, such as infrastructure for ubiquitous computing environments and frameworks for software service composition. He is supported by a Stanford Graduate Fellowship and an National Science Foundation Fellowship.

**Nathan Williams** received an MS degree in electrical engineering from Stanford University in 2001.

**Takashi Fukushima** is a research engineer at the Kobe Steel Ltd. Electronics Research Laboratory. He received BE and ME degrees in electronic engineering from Osaka University in 1991 and 1993. He has been with Kobe Steel Ltd. since 1993, where he has been engaged in the research and development of intelligent systems. He was a visiting scholar at the Computer Science Department of Stanford University during 2000 and 2001. His current research interests cover wireless/mobile computing and distributed systems.

**Hideki Yoshida** is a research scientist at the Corporate Research and Development Center of Toshiba Corporation and is a visiting scholar at the Computer Science Department of Stanford University. He received an MS degree in information science from the University of Tokyo. His research interests include mobile information access and network applications. He is a member of the ACM, the IEEE and the IPSJ.

**Vince Laviano** is a PhD candidate in the Computer Science Department at Stanford University. He received BS and MS degrees in computer science from George Mason University in 1995 and 1997. His primary research interests are in computer networks, distributed systems and operating systems, with his current work focusing on name-based multicast.

**Mary Baker** is an Assistant Professor in the Departments of Computer Science and Electrical Engineering at Stanford University. Her interests include operating systems, distributed systems, and mobile networking. She is now leading the development of the MosquitoNet mobile and wireless computing project and the Mobile People Architecture. Dr. Baker received a BA degree in mathematics in 1984 from the University of California at Berkeley, and a PhD in computer science in 1994 also from UC Berkeley. Dr. Baker is a recipient of an Alfred P. Sloan Research Fellowship, a Terman Fellowship, an NSF Faculty Career Development Award, and an Okawa Foundation grant.